

4장. 변수

👤 생성자	👤 한빈 이
🕒 생성 일시	@2024년 2월 16일 오후 11:00
🏷 태그	공부

“빨리 가는 유일한 방법은 제대로 가는 것이다.” - 클린코드의 저자, 로버트 C.마틴

4장. 변수

4.1 변수란 무엇인가? 왜 필요한가?

1. 메모리 주소를 통해 값에 직접 접근 → **치명적 오류** 발생 가능성 ↑

➡ 따라서, 자바스크립트는 개발자의 직접적인 메모리 제어를 허용 X

2. **변수(variable)** “협업과 품질 향상을 위해 변수는 첫아이 이름을 짓듯이 심사숙고”

: 하나의 값을 저장하기 위해 확보한 메모리 공간 자체 or 그 메모리 공간을 식별하기 위해 붙인 이름

➡ 간단하게, 값의 위치를 가리키는 상징적인 이름

3. **할당(assignment)(대입, 저장)** : 변수에 값을 저장하는 것

참조(reference) : 변수에 저장된 값을 읽어 들이는 것

4.2 식별자

1. 변수의 또 다른 이름, **식별자(identifier)** : 어떤 값을 구별해서 식별할 수 있는 고유한 이름

4.3 변수 선언

1. 변수 선언 : **var, let, const** 키워드 사용 (* var 키워드의 단점)

```
var score; // 변수 선언(변수 선언문)
```

↳ 아직 변수에 값 할당 X

변수 선언에 의해 확보된 메모리 공간은 비어있지않고,

확보된 메모리 공간에는 자바스크립트 엔진에 의해 **undefined**가 암묵적으로 할당되어 초기화 됨.

1-1. 초기화 단계를 거치지 않으면 확보된 메모리 공간에는 **쓰레기 값(garbage value)** 이 들어있다.

➡ 자바스크립트의 var 키워드는 암묵적으로 초기화를 수행하여 이러한 위험으로부터 안전

1-2. 선언하지 않은 식별자에 접근 → **ReferenceError(참조에러)** 발생

4.4 변수 선언의 실행 시점과 변수 호이스팅

1. [예제 04-05]

```
console.log(score); // undefined
```

```
var score; // 변수 선언문
```

⇒ 변수 선언문보다, 변수 참조하는 코드가 앞에 있음.

➡ 에러가 발생할 것 같지만, undefined가 출력됨.

이유는, 변수 선언이 소스코드 한 줄씩 순차적으로 실행되는 시점,
즉 런타임이 아니라 그 이전 단계에서 먼저 실행되기 때문.

변수 호이스팅

: 변수 선언문이 코드의 선두로 끌어 올려진 것처럼 동작하는 자바스크립트 고유의 특징

4.5 값의 할당

1. 변수에 값을 할당(대입, 저장)할 때 : 할당 연산자 =

2. [예제 04-06]

```
var score;    // 변수 선언
score = 80;   // 값의 할당
```

! 주의할 점 !

- 변수 선언, 값의 할당의 실행 시점이 다르다.
- **변수 선언**은 소스코드가 순차적으로 실행되는 시점인 런타임 이전에 먼저 실행되지만,
값의 할당은 소스코드가 순차적으로 실행되는 시점인 런타임에 실행된다.

3. `var score = 80;`

- 이와 같이 하나의 문장으로 표현해도,
자바스크립트 엔진은 변수의 선언과 값의 할당을 2개의 문으로 나누어 각각 실행한다.

4. [예제 04-10] - 자신에게 설명해보고, 실제로 실행해보기

```
console.log(score); // undefined
score = 80;         // 값의 할당
var score;          // 변수 선언
console.log(score); // ??
```

Q. 어떤 결과가 나올 것인가?

A. 값의 할당은 런타임에 실행되고, 변수 선언은 런타임 이전에 실행되기 때문에 **80**이 출력될 것.

```
console.log(score); // undefined
score = 80;        // 값의 할당
var score;         // 변수 선언
console.log(score);
이 코드의 결과를 알려주고 이유도 자세하게 알려줘
```

이한빈의 어시스턴트 GPT-4

이 코드를 실행하면 첫 번째 `console.log(score)`의 결과는 'undefined'이고, 두 번째 `console.log(score)`의 결과는 '80'입니다.

자세한 설명은 다음과 같습니다:

1. JavaScript에서는 변수 선언이 코드의 맨 위로 호이스팅(hoisting)되는 특징이 있습니다. 이는 변수가 선언된 위치와 상관없이 스크립트가 실행되기 전에 메모리에 할당되는 것을 의미합니다. 따라서 첫 번째 `console.log(score)`가 실행될 때, 변수 'score'는 이미 선언되었지만 아직 값이 할당되지 않았기 때문에 'undefined'를 반환합니다.
2. 이후 `'score = 80;'`라는 코드에서 'score' 변수에 값 '80'이 할당됩니다.
3. 마지막으로 두 번째 `console.log(score)`가 실행되는데, 이 때 'score' 변수에는 이미 '80'이라는 값이 할당되어 있으므로 '80'을 출력합니다.

따라서 이 코드의 최종 출력 결과는 'undefined'와 '80'입니다.

👍 🔄 📄 ✨ 문장 다듬기

4.6 값의 재할당

1. 값을 재할당할 수 없어서 변수에 저장된 값을 변경할 수 없다면, 변수가 아니라 **상수 (constant)**

(ES6에서 `const`를 사용한 변수는 재할당이 금지된다.)

2. 가비지 콜렉터(garbage collector) 란 ?

: 애플리케이션이 할당한 메모리 공간을 주기적으로 검사하여, 더 이상 사용되지 않는 메모리를 해제하는 기능

4.7 식별자 네이밍 규칙

1. 네이밍 규칙

규칙1. 특수문자를 제외한 문자, 숫자, 언더스코어(_), 달러 기호(\$)를 포함할 수 있다.

규칙2. 식별자는 특수문자를 제외한 문자, 숫자, 언더스코어(_), 달러 기호(\$)로 시작해야 한다.

숫자로 시작하는 것은 허용 X

규칙3. 예약어는 식별자로 사용할 수 없다.

2. 카멜 케이스(camelCase) : `var firstName;` - 주로 변수, 함수

스네이크 케이스(snake_case) : `var first_name;`

파스칼 케이스(PascalCase) : `var FirstName;` - 주로 생성자 함수, 클래스 이름

헝가리언 케이스(typeHungarianCase) : `var strFirstName;`

